TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**

BÀI TẬP LỚN MÔN PHÂN TÍCH THIẾT KẾ VÀ GIẢI THUẬT

# SẮP XẾP VÀ TÌM KIẾM

Giảng viên dạy lý thuyết: **TS. Nguyễn Chí Thiện**

Giảng viên dạy thực hành: **TS.Nguyễn Minh Tuấn**

Người thực hiện: **Hồ Phương Hiếu - 51303286**

Lớp : **13050303**

Khóa : **17**

THÀNH PHỐ HỒ CHÍ MINH, 2016

# LỜI CẢM ƠN

Em xin chân thành cảm ơn, thầy Nguyễn Chí Thiện đã hướng dẫn em phần lý thuyết môn Phân tích thiết kế và giải thuật, cảm ơn thầy đã ra đề tài “Sắp xếp và Tìm Kiếm” để em có thể tìm hiểu và hiểu sâu hơn các vấn đề về các loại giải thuật sắp xếp, và tìm kiếm. Đồng thời em xin chân thành cảm ơn thầy Nguyễn Minh Tuấn đã hướng dẫn em trong buổi thực hành để em có thể cài đặt các giải thuật trong thực tế.

# BÀI TẬP LỚN ĐƯỢC HOÀN THÀNH
# TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi và được sự hướng dẫn của TS Nguyễn Chí Thiện. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung bài tập lớn của mình.** Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày  tháng  năm*

*Tác giả*

*(ký tên và ghi rõ họ tên)*

Hồ Phương Hiếu

# PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

**Phần xác nhận của GV hướng dẫn**

_____

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày  tháng  năm

(ký và ghi họ tên)

**Phần đánh giá của GV chấm bài**

_____

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày  tháng  năm

(ký và ghi họ tên)

# TÓM TẮT

Đồ án gồm có hai phần : Sắp xếp và Tìm kiếm.Trong phần sắp xếp ta sẽ trình bày cụ thể từng thuật toán từ ý tưởng , cài đặt đến, phân tích độ phức tạp của từng giải thuật, so sánh tính ưu việt của các loại giải thuật, các mặt hạn chế để từ đó rút ra kết luận về hiệu quả của từng thuật toán. Trong  phần Tìm kiếm ta sẽ trình bày về các loại giải thuật tìm kiếm. Đầu tiên ta xét về giải thuật Brute-Force , đó là một giải thuậ cổ điển, tổng quát cho mọi trường hợp tìm kiếm. Nhưng vì nó tổng quát niên hiệu quả sẽ không cao do đó ta sẽ tìm kiếm những giải pháp thay thế cho nó.

# MỤC LỤC

# MỤC LỤC BẢNG



# MỤC LỤC BIỂU ĐỒ

# Phần I SẮP XẾP

## I) PHÂN TÍCH GIẢI THUẬT

### 1) Giải thuật Selection Sort

- Thiết kế:

Bước 1: i =1;

Bước 2: Tìm phần tử a[min] trong dãy hiện hành từ a[i] đến a[N]

Bước 3: Hoán vị a[min] với a[i]

Bước 4: Nếu I <= N-1 thì I = i+1;

Ngược lại: Dừng.

- Cài đặt:

```
public long[] SelectionSort(long[] a)
        {
            long min;
            for (long i = 0; i < a.Length; i++)
            {
                min = i;
                for (long j = i + 1; j < a.Length; j++)
                {
                    if (a[j] < a[min])
                    {
                        min = j;
                    }
                }
                if(min != i)
                swap(ref a[min], ref a[i]);
            }
            return a;
        }
```

### 2) Giải thuật Insertion Sort

- Thiết kế:

Bước 1: i = 2;

Bước 2: x = a[i]; Tìm vị trí pos thích hợp trong đoạn a[1] đến a[i-1] để chèn a[i] vào

Bước 3: Dời chổ các phần tử từ a[pos] đến a[i-1] sang phải 1 vị trí để dành chổ cho a[i]

Bước 4: a[pos] = x

Bước 5: i =i+1;

Nếu  i<=n: Lặp lại bước 2

Ngược lại dừng.

- Cài đặt:

```
public void InsertionSort(long[] a)
        {
            long pos, i;
            long x;
            for (i = 1; i < a.Length; i++)
            {
                x = a[i]; pos = i - 1;
                while ((pos >= 0) && (a[pos] > x))
                {
                    a[pos + 1] = a[pos];
                    pos--;
                }
                a[pos + 1] = x;
            }
        }
```

3) Giải thuật Bubble Sort

- Thiết kế:

Bước 1: i =1;

Bước 2: j = N ;

Trong khi  (j<i) thực hiện:

Nếu a[j] <a[j-1]: a[j]

Dời chổ các phần tử từ a[pos] đến a[i-1] sang phải 1 vị trí để dành chổ cho a[i]↔a[j-1]

j =j -1;

Bước 3: i = i+1; // lần xử lý kế tiếp

Nếu  i>n-1:Hết dãy. Dừng.

Ngược lại : Lặp lại bước 2.

- Cài đặt:

```
public void bubleSort(long[] a)
        {
            long i, j;
```

```
            for (i = 0; i < a.Length - 1; i++)
            {
                for (j = a.Length - 1; j > i; j--)
                    if (a[j] < a[j - 1])
                    {
                        long tam = a[j];
                        a[j] = a[j - 1];
                        a[j - 1] = tam;
                    }
            }
        }
```

4) Giải thuật Merge Sort

- Thiết kế:

Bước 1: tạo biến mid;

Bước 2: Nếu low < high

Tách dãy $a_1\ a_2\ a_{3\ldots}\ a_n$ thành các dãy con theo nguyên tắc luân phiên từ nhóm .

Bước 3: Trộn 2 mảng đã sắp xếp lại với nhau.

- Cài đặt

  - Merge

```
public void Merge(long[] a, long nb, long nc, long k, long n, long[]
b, long[] c)
        {
            long p, pb, pc, ib, ic, kb, kc;
            p = pb = pc = 0; ib = ic = 0;
            while (0 < nb && 0 < nc)
            {
                kb = (k > nb) ? nb : k;
                kc = (k > nc) ? nc : k;
                if (b[pb + ib] < c[pc + ic])
                {
                    a[p++] = b[pb + ib]; ib++;
                    if (ib == kb)
                    {
                        for (; ic < kc; ic++)
                        {
                            a[p++] = c[pc + ic];
                        }
                        pb += kb; pc += kc; ib = ic = 0;
                        nb -= kb; nc -= kc;
                    }
                }
                else
                {
```

```
                        a[p++] = c[pc + ic]; ic++;
                        if (ic == kc)
                        {
                            for (; ib < kb; ib++) a[p++] = b[pb + ib];
                            pb += kb; pc += kc; ib = ic = 0;
                            nb -= kb; nc -= kc;
                        }
                    }
                }
            }
```

- MergeSort

```
public void MergeSort(long[] a, long n)
        {
            long[] b = new long[n];
            long[] c = new long[n];
            long p, pb, pc;
            long i, k = 1;
            do
            {
                p = pb = pc = 0;
                while (p < n)
                {
                    for (i = 0; (p < n) && (i < k); i++)
                        b[pb++] = a[p++];
                    for (i = 0; (p < n) && (i < k); i++)
                        c[pc++] = a[p++];
                }
                Merge(a, pb, pc, k, n, b, c);
                k *= 2;
            } while (k < n);
        }
```

5) Giải thuật Quick Sort

(1) Giải thuật Quick Sort Middle

➢ Thiết kế:

Bước 1: Nếu left < right phân hoạch dãy $a_{left}$ $a_{right}$

Bước 2: Sắp xếp đoạn 1: $a_{left}$.. $a_j$

Bước 3: Sắp xếp đoạn 3: $a_i$.. $a_{right}$

➢ Cài đặt:

```
public void QuickSortMiddle(long[] a, long left, long right)
        {
            long i, j, x;
            x = a[(left + right) / 2];
            i = left; j = right;
            do
```

```
        {
            while (a[i] < x) i++;
            while (a[j] > x) j--;
            if (i <= j)
            {
                swap(ref a[i], ref a[j]);
                i++; j--;
            }
        } while (i < j);
        if (left < j)
            QuickSortMiddle(a, left, j);
        if (i < right)
            QuickSortMiddle(a, i, right);
    }
```

(2) Giải thuật Quick Sort First

- Thiết kế:

Bước 1: Nếu left ≥ right //dãy có ít hơn 2 phần tử

Kết thúc; //dãy đã được sắp xếp

Bước 2: Phân hoạch dãy $a_{left}$ … $a_{right}$ thành các đoạn: $a_{left}$.. $a_j$, $a_{j+1}$.. $a_{i-1}$, $a_i$.. $a_{right}$

Đoạn 1 ≤ x

Đoạn 2: $a_{j+1}$.. $a_{i-1}$ = x

Đoạn 3: $a_i$.. $a_{right}$ ≥ x

Bước 3: Sắp xếp đoạn 1: $a_{left}$.. $a_j$

Bước 4: Sắp xếp đoạn 3: $a_i$.. $a_{right}$

- Cài đặt:

```
public void QuickSortFirst(long[] a, long left, long right)
        {
            long i, j, x;
            x = a[left];
            i = left; j = right;
            do
            {
                while (a[i] < x) i++;
                while (a[j] > x) j--;
                if (i <= j)
                {
                    swap(ref a[i], ref a[j]);
                    i++; j--;
                }
            } while (i < j);
```

```
            if (left < j)
                QuickSortFirst(a, left, j);
            if (i < right)
                QuickSortFirst(a, i, right);
        }
```

(3) Giải thuật Quick Sort Random

- Thiết kế:
  - Bước 1: Nếu left < right phân hoạch dãy $a_{start}$ $a_{end}$
  - Bước 2: Sắp xếp đoạn 1$a_i$ $a_{end}$
  - Bước 3: Sắp xếp đoạn 3: $a_{start}$ $a_j$
- Cài đặt:

```
Random rd = new Random();
        public void QuickSortRandom(long[] a, long left, long right)
        {
            long i, j, x;
            int l = (int)left; int r = (int)right;
            int n = rd.Next(l, r);
            x = a[n];
            i = left; j = right;
            do
            {
                while (a[i] < x) i++;
                while (a[j] > x) j--;
                if (i <= j)
                {
                    swap(ref a[i], ref a[j]);
                    i++; j--;
                }
            } while (i < j);
            if (left < j)
                QuickSortRandom(a, left, j);
            if (i < right)
                QuickSortRandom(a, i, right);
        }
```

## II) Phân tích và dự đoán bằng lý thuyết

### 1) Giải thuật Selection sort

Để chọn được phần tử nhỏ nhất, ta cần duyệt qua n phần tử (tốn n-1 phép so sánh) và sau đó hoán vị nó với phần tử đầu tiên của dãy hiện hành. Để tìm phần tử nhỏ nhất tiếp theo, ta cần duyệt qua n-1 phần tử (tốn n-2 phép so sánh). Cứ như vậy, ta thấy ngay thuật toán sẽ tốn (n-1) + (n-2) + … + 1 = n(n-1)/2 = O(n2) phép so sánh. Mỗi lần duyệt, ta luôn phải hoán vị 1 lần (1 hoán vị tương đương với 3 phép gán), nghĩa là thuật toán sẽ tốn 3(n-1) + 3(n-2) + … + 3 = 3n(n-1)/2 = O(n2) phép gán.

Tổng kết lại, ta luôn có độ phức tạp của thuật toán Selection Sort thuộc O(n2) trong mọi trường hợp.

Ví du: dãy sau đã được sắp xếp : 1, 2, 3, 4, 5, 6, 7, 8, 9.

Dãy ngẫu nhiên các phần tử: 3, 6, 5, 7, 9, 4, 2, 1, 8.

Dãy được sắp xếp theo thứ tự ngược: 9, 8, 7, 6, 5, 4, 3, 2, 1.

Độ phức tạp không gian là O(1).

## 2) Giải thuật Insertion sort

(1) Trường hợp tốt nhất

Dãy ban đầu đã có thứ tự, không cần phải vào vòng lặp while.Với i chạy từ 2 đến n thì tổng số phép so sánh là n-1.

⇨ Độ phức tạp O(n)

Ví dụ: dãy sau đã được sắp xếp : 1, 2, 3, 4, 5, 6, 7, 8, 9.

(2) Trường hợp xấu nhất

Dãy ban đầu có thứ tự ngược. Vòng lặp for chạy n-1 lần, vòng lặp while chạy n-1 lần.

⇨ Độ phức tạp O($n^2$)

✓ Độ phức tạp không gian là O(1).

Ví dụ: dãy sau được sắp xếp theo thứ tự ngược: 9, 8, 7, 6, 5, 4, 3, 2, 1.

(3) Trường hợp trung bình

Có khoảng chừng (i-1)/2 so sánh đượct thực thi trong vòng lặp. Do đó, trong trường hợp trung bình tổng số lần so sánh là (N-1)/2+(N-2)/2+…1/2=N(N-1)/4 = O($N^2$)

⇨ Độ phức tạp O($n^2$)

Ví dụ dãy sau: 1, 2, 3, 4, 5, 9, 8, 6, 7.

## 3) Giải thuật Bubble Sort

(1) Trường hợp tốt nhất

Dãy ban đầu đã có thứ tự, số phép so sánh là n-1 , không tốn một phép hoán vị nào.

⇨ Độ phức tạp O($n^2$)

Ví dụ: dãy sau đã được sắp xếp : 1, 2, 3, 4, 5, 6, 7, 8, 9.

(2) Trường hợp xấu nhất

Dãy ban đầu có thứ tự ngược.

⇨ Độ phức tạp O($n^2$)

Ví dụ: dãy sau được sắp xếp theo thứ tự ngược: 9, 8, 7, 6, 5, 4, 3, 2,

✓ Độ phức tạp không gian là O($n^2$)

(3) Trường hợp trung bình

Dãy ban đầu có thứ tự ngẫu nhiên.

⇨ Độ phức tạp O($n^2$)

Ví dụ:Dãy ngẫu nhiên các phần tử: 3, 6, 5, 7, 9, 4, 2, 1, 8.

## 4) Giải thuật Merge Sort

Ta thấy ngay số lần lặp của bước 2(phân phối) và bước 3(trộn) bằng log2n. Ta cũng thấy rằng chi phí thực hiện bước 2 và bước 3 tỉ lệ thuận với n. Như vậy, ta có thể ước tính chi phí thực hiện của giải thuật Merge Sort thuộc O(nlog2n).

Ta nhận thấy rằng giải thuật làm việc một cách cứng nhắc, không tận dụng được tính thứ tự một phần của dãy ban đầu. Do đó, trong mọi trường hợp độ phức tạp là như nhau. Đây là một nhược điểm của phương pháp trộn trực tiếp.

Ví du: dãy sau đã được sắp xếp : 1, 2, 3, 4, 5, 6, 7, 8, 9.

Dãy ngẫu nhiên các phần tử: 3, 6, 5, 7, 9, 4, 2, 1, 8.

Dãy được sắp xếp theo thứ tự ngược: 9, 8, 7, 6, 5, 4, 3, 2, 1.

⇨ Độ phức tạp không gian là O(n).

## 5) Giải thuật Quick sort

(1) Trường hợp tốt nhất

Trường hợp tốt nhất xảy ra khi nỗi lần phân hoặc chia tập tin ra thành hai phần bằng nhau. Nó làm số lần so sánh của Quick Sort thỏa mãn hệ thức truy hồi:

$C_N = 2C_{N/2} + N$

Số hạng $2C_{N/2}$ là chi phí của việc sắp xếp thứ tự hai nữa tập tin và N là chi phí của việc xét từng phầnt ử khi phan hoạch lần đầu. Do đó $C_N \approx N\ lg_N$

⇨ Độ phức tạp O($N\ lg_N$)

(2) Trường hợp xấu nhất

Tập tin đã có thứ tự . Khi đó phần tử thứ nhất sẽ đòi n so sánh để nhận ra rằng nó nên đứng ở vị trí thứ nhất . Hơn nữa sau đó phân đoạn bên trái là rỗng và phân đoạn bên phải là n-1 so sánh để nhận ra rằngnó nên ở đúng vị trí thứ hai. Và cứ tiếp tục thế. Như vậy tổng số lần so sánh là:

N+(N-1)+ ….+2+1 = N(N+1)/2 = ($N^2$ +N)/2 = O($N^2$)

⇨ Độ phức tạp O($n^2$)

$C_0 + C_1 + \cdots + C_{N-1}$ thì giống hệt $C_{N-1}$ + $C_{N-2}$ + … + $C_0$, nên ta có

$C_N$ = (N+1) + (1/N)$\sum_1^N 2C_{k-1}$

Ta có thể loại trừ đại lượng tính tổng bằng cách nhân cả hai vế với N và rồi trừ cho công thức hân vơi N-1: N$C_N - (N-1)C_{N-1}$ = N(N+1) – (N-1)N+2$C_{N-1}$. Từ đó ta được N$C_N$ = $(N+1)C_{N-1}$ + 2N

Chia cả hai vế với N(N+1) ta được hệ thức truy hồi:

$C_N$/(N+1) = $C_{N-1}$/N + 2/(N+1)

= $C_{N-2}$/(N-1) + 2/N + 2(N+1)

= $C_2$/3 + $\sum_3^N 2/(k+1)$

$C_N$/(N+1) $\approx 2\sum_1^N \frac{1}{k} \approx 2\int_1^N 1/x$dx = 2lnN

⇨ $C_N \approx 2NlnN$

Vì ta có :

lnN = ($log_2$).($log_e$ 2 = 0.69lgN

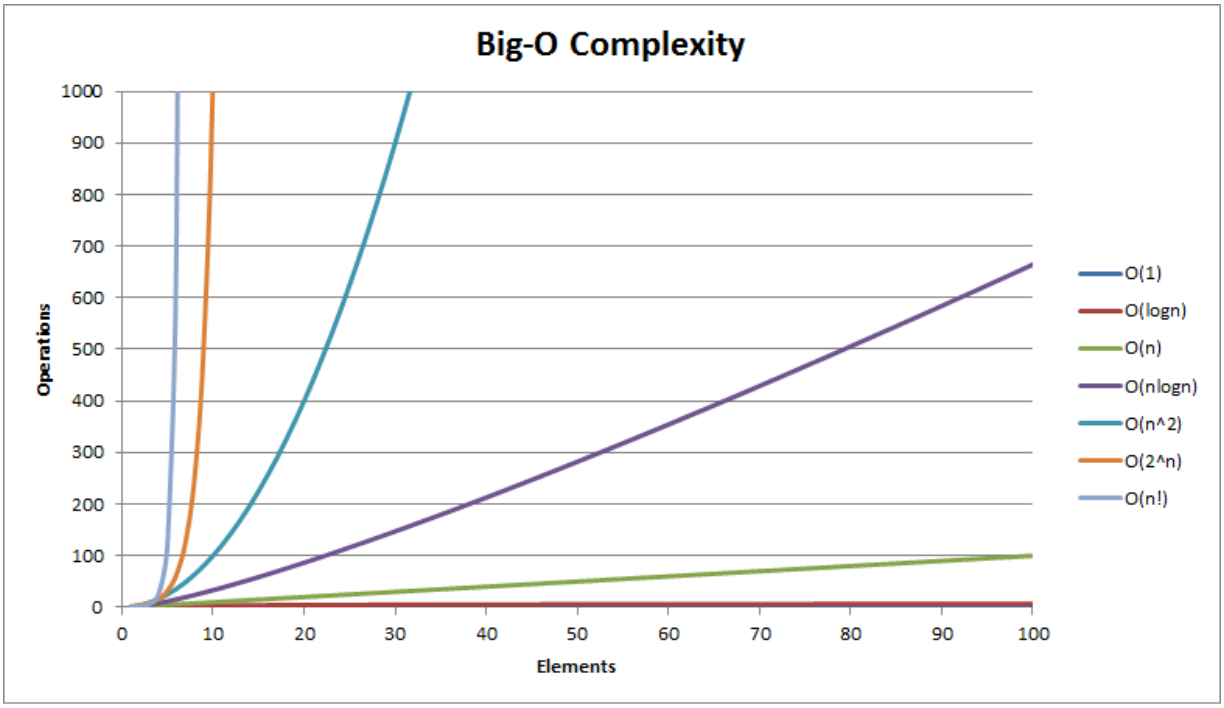
2NlnN ≈ 1.38NlgN

⇨ Tổng số so sánh trung bình của Quick Sort chỉ khoảng chừng 38% cao hơn trong trường hợp tốt nhất.

Sau những phân tích lý thuyết ta có những nhận xét sau đây:

- Giải thuật Selection Sort có độ phức tạp thời gian về tất cả các trường hợp cao nhất ($O(n^2)$)
- Các giải thuật Merge Sort có độ phức tạp thời gian khá lớn ($O(n\log(n))$ nhưng lại máy móc và không tận dụng được những phần tử đã sắp xếp

Biểu đồ sau đây có thể cho ta thấy độ phức tạp thời gian biến thiên như thế nào với kích thước dữ liệu tăng dần:




Với các kết quả lý thuyết, ta có thể dự đoán được:

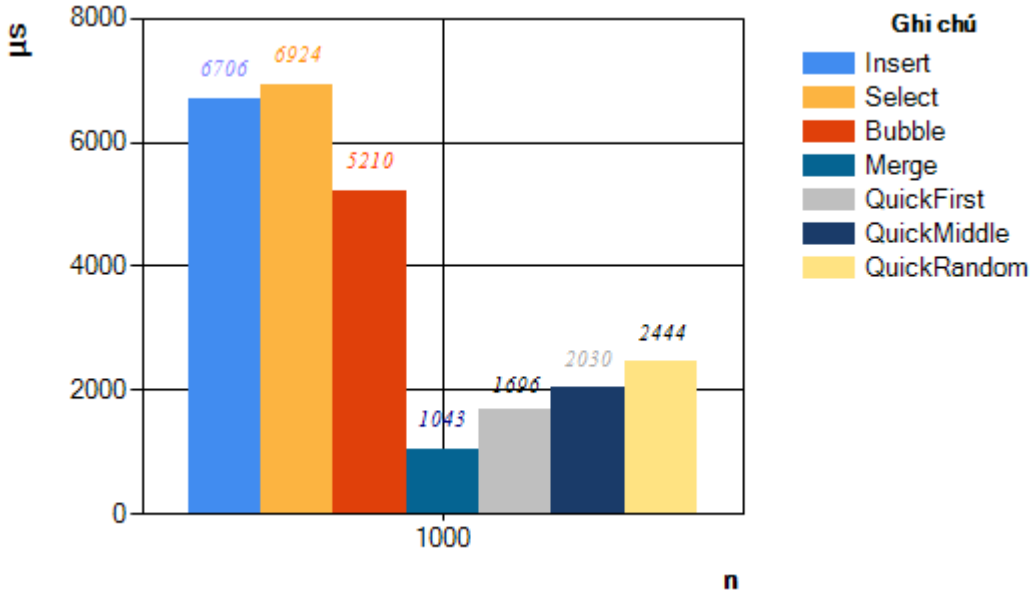
- Giải thuật có độ phức tạp thời gian thấp nhất: Quick Sort First
- Giải thuật có độ phức tạp thời gian cao nhất: Selection Sort

## III) Tạo dữ liệu và thiết lập môi trường thực nghiệm

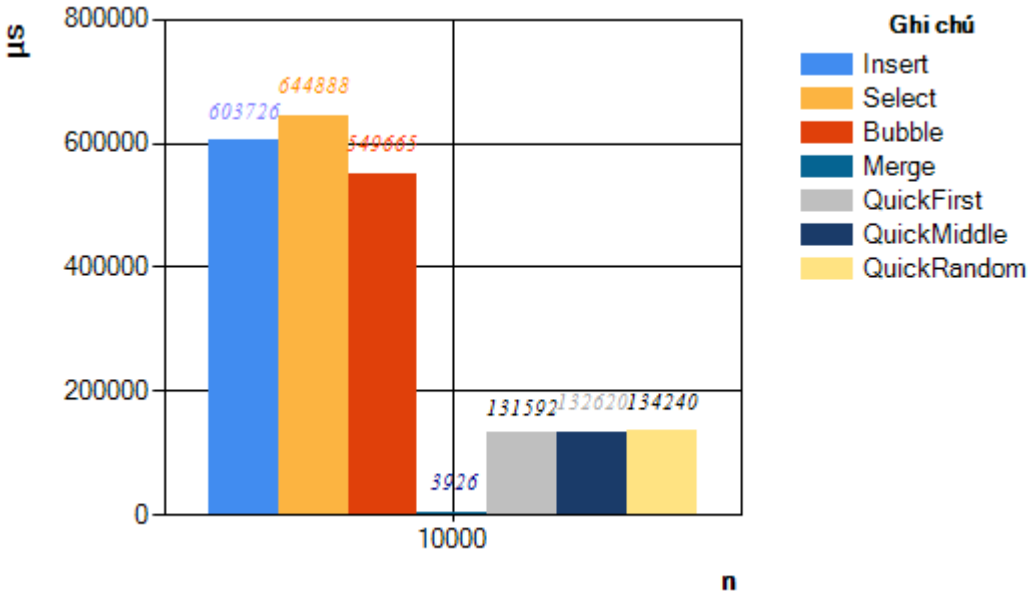
### 1) Cấu hình thực nghiệm

- CPU : i5 –3337U up to 1.8 GHz.
- RAM 6GB - DDR3 - 1600Mhz.
- Hệ điều hành: Windows 10 64bit
- IDE : Visual studio 2013 Ultimate.
- Ngôn ngữ C#.

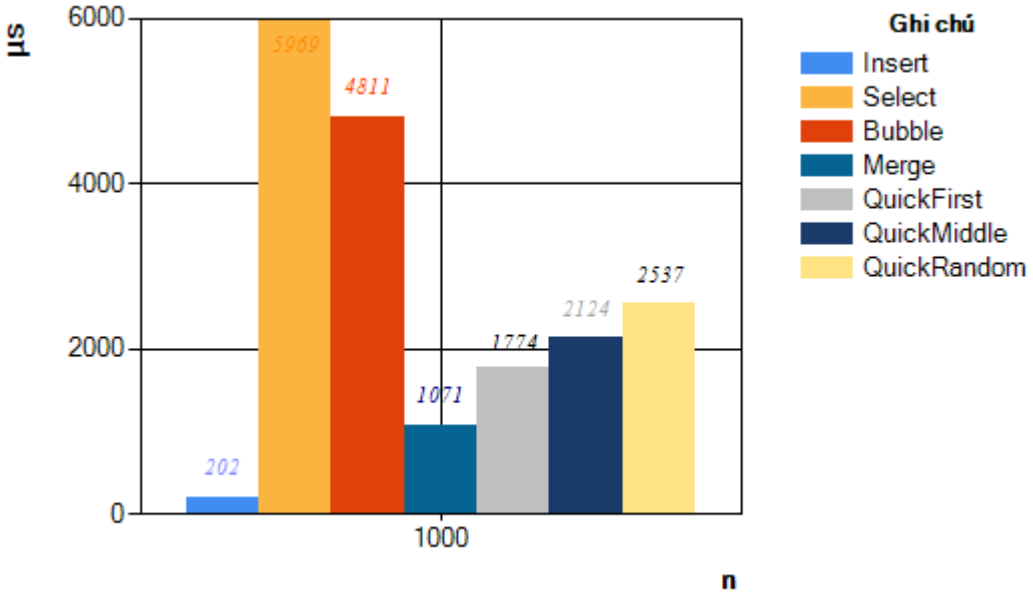
2) Biểu đồ phân tích



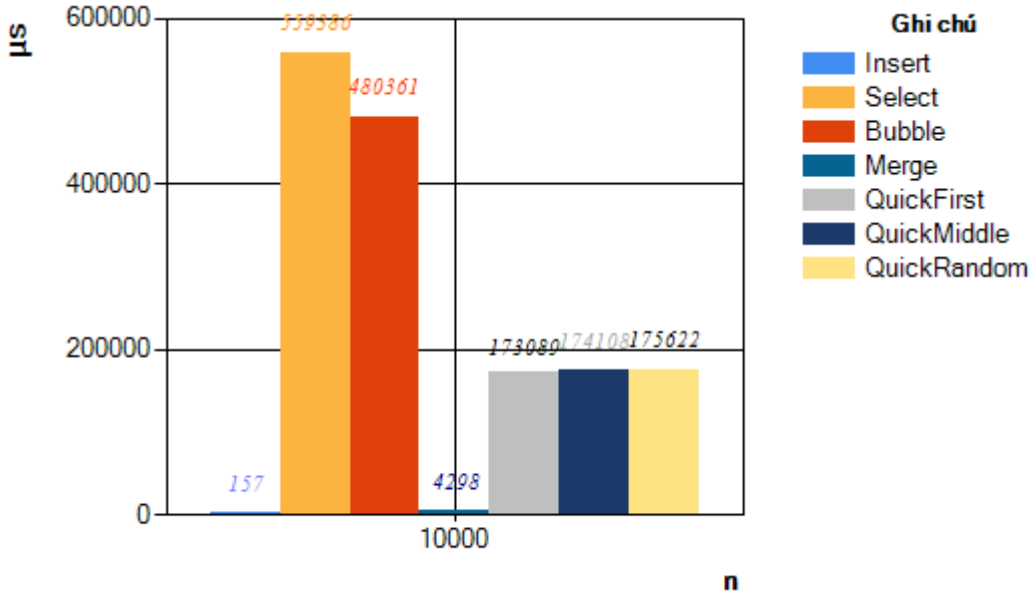

*Biểu đồ 1 Sắp xếp số lượng phần tử 1000 với mảng giảm dần*



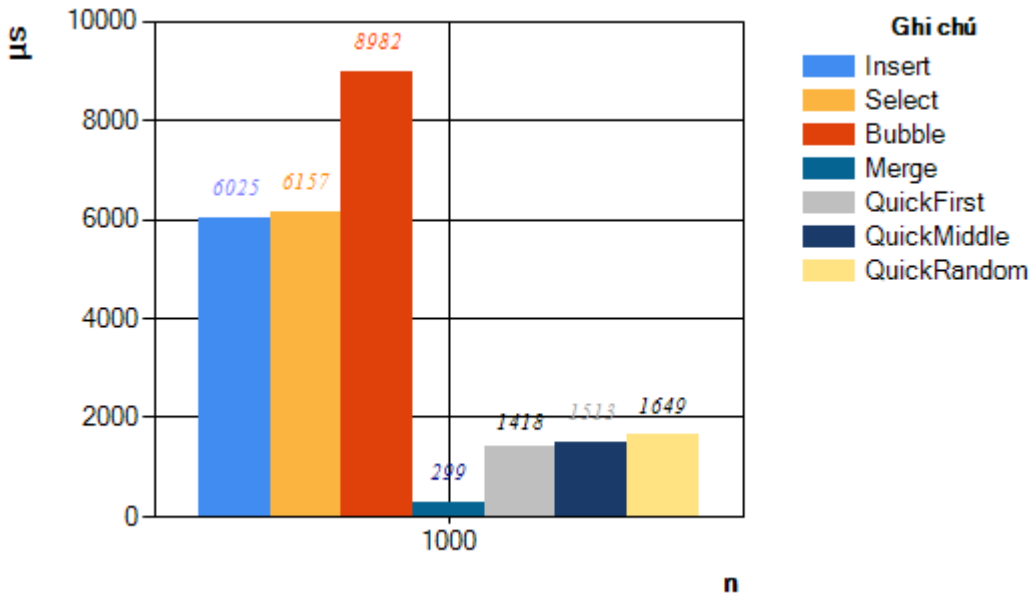

*Biểu đồ 2 Sắp xếp số lượng phần tử 10.000 với mảng giảm dần*



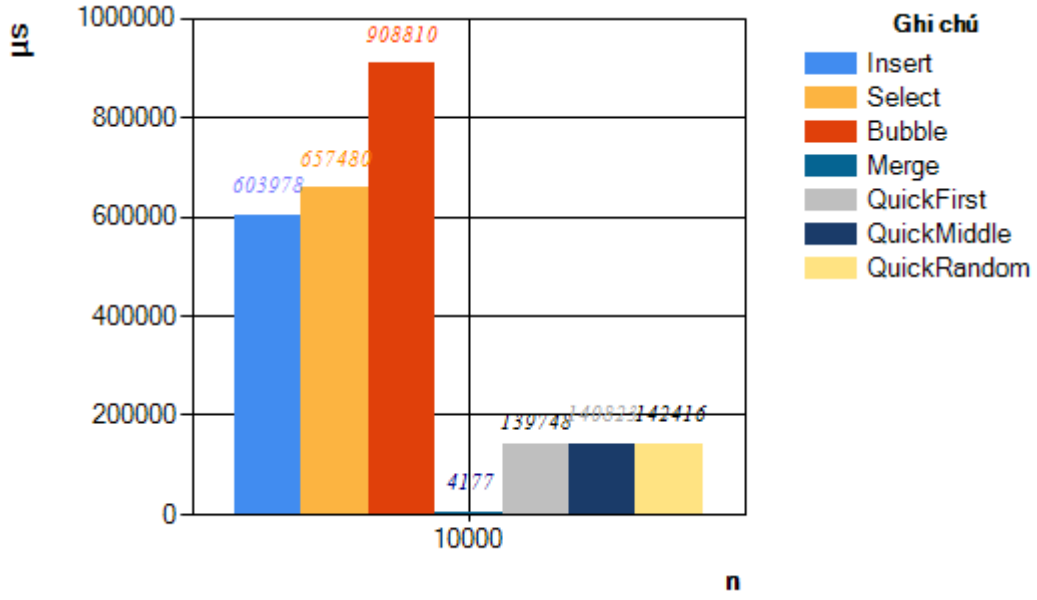

*Biểu đồ 3 Sắp xếp số lượng phần tử 1000 với mảng tăng dần*




*Biểu đồ 4 Sắp xếp số lượng phần tử 10.000 với mảng tăng dần*




*Biểu đồ 5 Sắp xếp số lượng phần tử 1000 với mảng ngẫu nhiên*




*Biểu đồ 6 Sắp xếp số lượng phần tử 10.000 với mảng ngẫu nhiên*

3) Bảng số liệu cụ thể

| Kiểu dữ liệu / Thời gian(micro s) | Xếp tăng dần | Xếp ngẫu nhiên | Xếp giảm dần |
|---|---|---|---|
| Insertion Sort | 202 | 6025 | 6706 |
| Selection Sort | 5969 | 6157 | 6924 |
| Bubble Sort | 4811 | 8982 | 5210 |
| Merge Sort | 1071 | 299 | 1043 |
| QuickSort First | 1774 | 1418 | 1696 |
| QuickSort Middle | 2124 | 1513 | 2030 |
| QuickSort Random | 2537 | 1649 | 2444 |

*Bảng I-1 Số Lượng Phần Tử :1000*

| Kiểu dữ liệu / Thời gian(micro s) | Xếp tăng dần | Xếp ngẫu nhiên | Xếp giảm dần |
|---|---|---|---|
| Insertion Sort | 157 | 603978 | 603726 |
| Selection Sort | 559380 | 657480 | 644888 |
| Bubble Sort | 480361 | 908810 | 549665 |
| Merge Sort | 4298 | 4177 | 3926 |
| QuickSort First | 173080 | 139748 | 131592 |
| QuickSort Middle | 174108 | 140823 | 132620 |
| QuickSort Random | 175622 | 142416 | 134240 |

*Bảng I-2 Số lượng phần tử :10000*

4) Nhận xét thời gian chạy thực tế của các giải thuật

Theo như Biểu đồ 1,2 và 3 ta rút ra nhận xét

Giải thuật Selection Sort cho thấy sự kém hiệu quả so với các giải thuật còn lại theo kiểu sắp xếp tăng dần, cụ thể đối với số lượng 1000 phần tử thì giải thuật Selection Sort chậm hơn khoảng 1.5 lần so với giải thuật nhanh xếp sau nó là Bubble Sort, đối với số lượng 10.000 phần tử thì nó chạy chậm hơn khoảng 0.7 lần so với giải thuật Bubble Sort (giải thuật chạy chậm kế tiếp).

Đối với giải thuật Bubble Sort thì cho thấy thời gian chạy chậm hơn so với các giải thuật còn lại. Cụ thể đối với số lượng phần tử sắp xếp là 1000 và theo mảng ngẫu nhiên thì giải thuật Bubble Sort chạy chậm gấp khoảng 1.5 lần so với giải thuật chạy chậm sau nó.

Đối với giải thuật Merge Sort thì trung bình thời gian chạy tốt nhất đối với mọi kiểu sắp xếp mảng so với các giải thuật còn lại. Cụ thể đối với mảng có kiểu ngẫu nhiên và số lượng phần tử khoảng 1000 thì giải thuật Merge Sort tỏ ra rất hiệu quả, Merge Sort chạy nhanh hơn khoảng 4.5 lần so với giải thuật chạy nhanh sau nó là QuickSort First, nhưng đối với số lượng phần tử khoảng 10.000 thì giải thuật lại tỏ ra hiệu quả rất nhiều lần, cụ thể nó chạy nhanh gấp khoảng 30 lần so với giải thuật chạy nhanh sau nó là QuickSort First.

Đối với giải thuật Insertion Sort thì nó tỏ ra rất hiệu quả đối với mảng tăng dần, cụ thể nó chạy nhanh gấp khoảng 27 lần so với giải thuật Merge Sort (với 10.000 phần tử) và nhanh gấp khoảng 5.3 lần so với giải thuật Merge Sort (với 1000 phần tử).

# Phần II TÌM KIẾM

## I) Tìm kiếm tuần tự

### 1) Mô tả

Duyệt lần lượt từng phần tử trong mảng cho đến khi tìm thấy giá trị mong muốn

Input : mảng a[n] có n phần tử, khóa tìm kiếm k.

Output: vị trí phần tử có khóa bằng k.

### 2) Thiết kế

Algorithms **Sequential Search**(arr A, var k)

1. Chạy vòng lặp i từ 0 đến A.lenght
2. Kiểm tra A[i] = k hay không
3. Nếu có thì return i

### 3) Phân tích xác định độ phức tạp

| Trường hợp | Số lần so sánh | Giải thích |
|---|---|---|
| Tốt nhất | 1 | Phần tử cần tìm kiếm nằm ở đầu mảng |
| Trung bình | n | |
| Xấu nhất | n | Không tìm thấy phần tử trong mảng hoặc phần từ nằm ở cuối mảng |

*Bảng II-1 Bảng phân tích độ phức tạp của giải thuật tìm kiếm tuần tự*

## II) Tìm kiếm nhị phân

Có nhiều giải thuật tìm kiếm tốt hơn giải thuật Sequential Search , điển hình ở đây là giải thuật "Tìm kiếm nhị phân( Binary Search) " .

Nhưng để thực hiện được giải thuật "Tìm kiếm nhị phân" thì không gian tìm kiếm phải được sắp xếp trước theo khóa tìm kiếm.

(1) Mô tả

Input : mảng a[left. . . right] được sắp theo khóa tăng dần, khóa tìm kiếm k.

Output: vị trí phần tử có khóa bằng k.

Giải thuật tìm kiếm nhị phân ( Binary Search):

(2) Thiết kế

1. left =1; right = N;
2. mid = (left +right)/2;
3. So sánh a[mid] với x có 3 khả năng:
4. a[mid] =x; Tìm thấy.Dừng.
5. a[mid] > x ;right = mid - 1;
6. a[mid] <x; left = mid + 1;
7. Nếu left <= right;
8. Lặp lại bước 2
9. Ngược lại: Dừng;

- Cài đặt

```
   BinarySearch ( Array a , int x, int n)
1. left = 0, right = n-1, mid;
2. Respeat loop( left <= right)
3. mid = (left + right) div 2;
4. if(a[mid] >x) right = mid -1;
5. else if(a[mid] <x) left = mid +1;
6. else return mid;
7. if(left > right)
8. return -1;
```

- Phân tích và xác định độ phức tạp

| Trường hợp | Số lần so sánh | Giải thích |
|---|---|---|
| Tốt nhất | 1 | Phần tử đứng giữa mảng có giá trị x |
| Xấu nhất | $log_2 n$ | Không có x trong mảng |
| Trung bình | $log_2 n/2$ | Giả sử xác xuất các phần tử trong mảng nhận giá trị x là như nhau |

⇨ Với bảng phân tích ở trên giải thuật tìm kiếm nhị phân có độ phức tạp tính toán cấp n: T(n) = O($log_2 n$).

## III) Cây nhị phân tìm kiếm

### 1) Mô tả

Cây tìm kiếm ứng với n khóa k1,k2,….kn là cây nhị phân mà mỗi nút đều được gán một hóa sao cho với mỗi mỗi nút k:

Mọi khóa trên cây con trái đều nhỏ hơn khóa trên nút k

Mọi khóa trên cây con phải đều lớn hơn khóa trên nút k

2) Thiết kế

**Algorithms Search(var bst, k)**

```
   LPTNODE searchNode(TREE Root,  Data x)
1. LPNODE p = Root;
2. while (p != NULL)
3. if(x == p->Key)
4. return p;
5. if(x < p->Key)
6. p = p->pLeft;
7. if(x > p->Key)
8. p = p->pRight;
9. return NULL;
```

3) Phân tích và đánh giá độ phức tạp

Thao tác duyệt cây trên cây nhị phân tìm kiếm hoàn toàn giống như trên cây nhị phân. Chỉ có một lưu ý nhỏ là khi duyệt theo thứ tự giữa, trình tự các nút duyệt qua sẽ cho ta một dãy các nút theo thứ tự tăng dần của khóa.

Tìm một phần tử x trong cây

Dễ dàng thấy rằng số lần so sánh tối đa phải thực hiện để tìm phần tử X là h, với h là chiều cao của cây. Như vậy thao tác tìm kiếm trên CNPTK có n nút tốn chi phí trung bình khoảng O(log2n) .

4) Đánh Giá

Trên cây nhị phân tìm kiếm đều có độ phức tạp trung bình O(h), với h là chiều cao của cây. Trong trong trường hợp tốt nhất, cây nhị phân tìm kiếm có n nút sẽ có độ cao h = log2(n). Chi phí tìm kiếm khi đó sẽ tương đương tìm kiếm nhị phân trên mảng có thứ tự. Tuy nhiên, trong trường hợp xấu nhất, cây có thể bị suy biến thành một danh sách liên kết (khi mà mỗi nút đều chỉ có 1 con trừ nút lá). Lúc đó các thao tác trên sẽ có độ phức tạp O(n). Vì vậy cần có cải tiến cấu trúc của cây nhị phân tìm kiếm để đạt được chi phí cho các thao tác là log2(n).

| Trường hợp | Số lần so sánh | Giải thích |
|---|---|---|
| Tốt nhất | 1 | Phần tử cần tìm kiếm là nút gốc |
| Trung bình | $\log_2(n)$ | |
| Xấu nhất | n | Mỗi nút đều chỉ có 1 con trừ nút lá |

# TÀI LIỆU THAM KHẢO

Slide: Bài giảng phân tích thiết kế và giải thuật, thầy Nguyễn Chí Thiện, giảng viên khoa CNTT, đại học Tôn Đức Thắng.

Slide :Bài giảng của TS. Nguyễn Minh Tuấn, giảng viên khoa CNTT, đại học Tôn Đức Thắng.

Website: https://en.wikipedia.org/wiki/Sorting_algorithm

https://sites.google.com/site/nguoisaigonblog/giai-thuat-sap-xep-tinh-te

https://voer.edu.vn/m/thuat-toan-sap-xep/ff96502c

https://voer.edu.vn/c/phan-tich-thoi-gian-thuc-hien-thuat-toan/60bbf7d3/a7204439

Ebook https://uet.vnu.edu.vn/tltk/Learning/File_PDF/Cautrucdulieu.pdf